CSc 210: Software Development
Section 10: A-maze-ing Shapes
October 30th, 2017

Everyone loves the challenge of a corn maze, right? Well, maybe not. But in the spirit of fall, we will be implementing our own corn maze program! Our program will read in an ascii version of a maze and display it on the screen using JavaFX. We will then be able to traverse our maze by entering commands on the terminal. Let's get started.

Part I: Drawing our maze

1. Our first task will be to read in an ascii version of a maze as a 2d array and render that array onto the screen. First, open a file in your editor of choice called Drawing.java and create a class called Drawing that extends Application. Create a main method that calls launch and also override launch as we have seen in class. You should have at least the following import statements:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.Group;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
```

2. Look on the section page on the course website. There is code posted for a method that will read in an ascii maze from a file called maze.txt. There is also an example maze.txt file. This method returns a two dimensional character array that has the following properties:
    - If the character held in a position of the array is a '*', it is a border we can not pass through.
    - If it is a "S", that is the start point of the maze.
    - If it is a "E", that is the end point of the maze.
    - If the character is a space, it is an empty position we can pass through.

   Make a call to this method inside of the launch method to get the character array maze representation.

3. Now we know what our maze looks like, so let's draw it on the screen! We know how wide and tall our maze is, so we can create a Canvas object that has width by height many squares for us to draw our maze on. Let's make each square 25 by 25 pixels in size. How would we create a Canvas object that is appropriately sized?

4. Next, you should get the GraphicsContext from your canvas by calling getGraphicsContext2D(). Using this and the maze character array, draw out the maze onto your canvas. Each x,y position in the maze array should correlate to the square x,y on your screen. Follow the following rules:
    - If the character held in a position of the array is a '*', fill that square with yellow (in the spirit of a corn maze!).
    - If it is a "S", fill it with a blue triangle (our player moving through the maze).
    - If it is a "E", fill it with green (the final position).
    - If the character is a space, it is an empty position we can pass through, so do not fill it with anything.

    For drawing squares (rectangles), you will probably want to use the method in graphics context called fillRectangle(). For drawing the triangle, you probably want to use fillPolygon(). There is an example of what should be outputted with the example input on the sections page.

5. Once all the shapes are drawn, create a Group object, add the Canvas to the Group's children, set the scene of the primary stage, and display your window by calling show() on the primary stage.

6. If your program seems to be generating output on the screen that is similar to the example image, then you have completed the first part of this program! Now, we will read in commands to traverse the maze.

Part II: Command REPL

We will now create a small REPL to read in commands from standard input to traverse the maze. Note that we are not able to simultaneously have a GUI open and to read input on one thread of execution. So, we will have to start the execution of our REPL on its own thread.

1. To do this, first open a file called CommandREPL.java and create a class called CommandREPL. This class should extend Thread. You must implement the

public void run() method inherited from Thread. This is the method that will be run when execution of this thread is started.

2. The run method should be where you read input in. So, create a scanner and read in line by line. The possible commands you will read in are LEFT, RIGHT, UP, and DOWN, which each correspond to moving that direction in the maze. Add functionality to read and parse these commands.

3. Now, our REPL class is going to need some additional information to operate correctly. You should create a constructor that takes in a maze array, starting x and y positions of the player, and a Canvas GraphicsContext. From the launch method in Drawing, you should create a ShapeREPL class and pass in these parameters.

4. Now that we have the needed information, you should add functionality upon receiving a move command. When receiving a command, you should
    a. Check they are moving to a valid position. If not, dont do anything.
    b. Figure out what direction they are moving.
    c. Move the player triangle to that new position on the screen (using the GraphicsContext).
    d. Erase the old player triangle on the screen (using the GraphicsContext).

5. The final step is to add a call to start() on our ShapeREPL we defined in the Drawing class. We must call the start method to actually start the execution of our ShapeREPL thread. Place this call right before you call show() on your primaryStage.

6. If done correctly, you should have a working maze game!